In [ ]:

```
'''

Let's further explore the functionality of the pdutils-dfavg method using the

Titanic dataset. The dfavg method is used to return an individual estimate or

a series of approximations for the imputation of missing values in the designated

column of the source dataframe.

'''
```

In [89]:

```
'''

Let's start with importing all the necessary modules in the first cell. Once the modules

are imported, the cell output confirms with a message that all imports have been imported!

'''

import numpy as np
import pandas as pd
import warnings
from pdutils.imputation.dfavg import dfavg

warnings.simplefilter("ignore")

print('All imports have been imported!')
```

```
All imports have been imported!
```

In [90]:

```
'''

Load the Titanic dataset from Seaborn.

'''
titanic = sns.load_dataset('titanic')
titanic.head()
```

Out[90]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	ali
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	ı
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	y
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	y
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	y
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	ı

In [84]:

```
'''

From the Jupyter notebook on the dfplot method with Titanic data, it appeared that some

columns may be redundant. We will use the pdutils-dforder method to select and order

specific columns in the dataframe.

'''

from pdutils.tidying.dforder import dforder

dfz = titanic.dforder(columns=['age', 'sex', 'who', 'sibsp', 'parch', 'embarked',
                               'pclass', 'deck', 'fare', 'survived'])
print(dfz.shape)
print(dfz.columns)
dfz.head(10)
```

```
(891, 10)
Index(['age', 'sex', 'who', 'sibsp', 'parch', 'embarked', 'pclass', 'deck',
       'fare', 'survived'],
      dtype='object')
```

Out[84]:

	age	sex	who	sibsp	parch	embarked	pclass	deck	fare	survived

0	22.0	male	man	1	0	S	3	NaN	7.2500	0
1	38.0	female	woman	1	0	C	1	C	71.2833	1
2	26.0	female	woman	0	0	S	3	NaN	7.9250	1
3	35.0	female	woman	1	0	S	1	C	53.1000	1
4	35.0	male	man	0	0	S	3	NaN	8.0500	0
5	NaN	male	man	0	0	Q	3	NaN	8.4583	0
6	54.0	male	man	0	0	S	1	E	51.8625	0
7	2.0	male	child	3	1	S	3	NaN	21.0750	0
8	27.0	female	woman	0	2	S	3	NaN	11.1333	1
9	14.0	female	child	1	0	C	2	NaN	30.0708	1

In [16]:

```
'''

Let's use the dfsummarize method in comprensive mode to summarize the

reduced data set.

'''

from pdutils.summary.dfsummarize import dfsummarize

mtab, _ = dfz.dfsummarize(date_col_ls=[], num_col_ls=[], check_bool=False,
                          stats=False, to_sci_no=False, show_null_index_col=True)
mtab
```

```
*** Finding and replacing potential missing value variants in the dataframe...
*** The dataframe has no missing value variants of the type: ['NaN', 'nan', 'unknown', 'NA', 'na', 'N/A',
'M', '', ' '].
    Dataframe shape (r, c): (891, 10)
    Features (cols) list  : ['age', 'sex', 'who', 'sibsp', 'parch', 'embarked', 'pclass', 'deck', 'fare',
'survived']
    Missing value count   : 867
    All null row indexes  : [ 0   2   4   5   7   8   9 12 13 14 15 16 17 18 19 20 22 24 25 26]...

*** Searching data for missing values...
*** Eliminating missing values...
*** Row count has dropped from 891 to 182 due to null removal.
*** Post-row-removal null count across columns is 0.

Legend   :
   C-Type: Column data type
   V-Type: Column value data type
   MLS   : Maximum length of [column values converted into] string data type
   NIL   : Null index list
   SCIL  : Search column index list (list of col-specific row indexes that match the search criterion)
```

Out[16]:

	Feature	C-Type	V-Type	MLS	#NonNull	#Unique	#Null	NIL
0	age	float64	[float64]	4	714	63	177	[5, 17, 19, 26, 28, 29, 31, 32, 36, 42, 45, 46...
1	deck	category	[float, str]	3	203	7	688	[0, 2, 4, 5, 7, 8, 9, 12, 13, 14, 15, 16, 17, ...
2	embarked	object	[float, str]	3	889	3	2	[61, 829]
3	fare	float64	[float64]	8	891	93	0	[]
4	parch	int64	[int64]	1	891	4	0	[]
5	pclass	int64	[int64]	1	891	3	0	[]
6	sex	object	[str]	6	891	2	0	[]
7	sibsp	int64	[int64]	1	891	4	0	[]
8	survived	int64	[int64]	1	891	2	0	[]
9	who	object	[str]	5	891	3	0	[]

In [24]:

```
print(dfz.groupby(['deck', 'pclass'])['fare'].median().unstack().fillna(0))
```

```
pclass        1        2          3
deck
```

```
A       35.500   0.000   0.00000
B       80.000   0.000   0.00000
C       83.475   0.000   0.00000
D       75.250  13.000   0.00000
E       55.000  11.425  12.47500
F        0.000  26.000   7.65000
G        0.000   0.000  13.58125
```

In [14]:

```python
'''

Note that both `age` and `deck` columns have multiple missing values.

Let's explore further using native Pandas functions and try to develop
basic heuristics. The `who' column gives an indication of passenger type
by gender and age, and comprises no missing values.

Based on code execution below, a potential set of heuristics for `age`
imputation can be devised as follows:
- if who=='child', age=5
- if who!='child', age=30

For `deck` imputation, we may need a more complex set of heuristics:
- if pclass==3 and fare>13, deck='G'
- if pclass==3 and fare<11, deck='F'
- if pclass==3 and fare>=11, deck='E'
- if pclass==2 and fare>20, deck='F'
- if pclass==2 and fare<13, deck='E'
- if pclass==2 and fare>=13 and fare<=20, deck='D'
- if pclass==1 and fare<50, deck='A'
- if pclass==1 and fare>50 and fare<60, deck='E'
- if pclass==1 and fare>=60 and fare<80, deck='D'
- if pclass==1 and fare>=80 and fare<83, deck='B'
- if pclass==1 and fare>=83, deck='C'

'''

print('who associated with median age            :')
print(dfz.groupby('who')['age'].median())
print('who associated with median fare           :')
print(dfz.groupby('who')['fare'].median())

print('pclass type(s) associated with deck       :')
print(dfz.groupby('deck')['pclass'].unique())
print('Passenger count linked with pclass by deck:')
print(dfz.groupby('deck')['pclass'].value_counts().unstack().fillna(0))
print('The median fare specific to pclass by deck:')
print(dfz.groupby(['deck', 'pclass'])['fare'].median().unstack().fillna(0))
```

```
who associated with median age            :
who
child     5.0
man      30.0
woman    30.0
Name: age, dtype: float64
who associated with median fare           :
who
child    26.25
man       9.50
woman    23.25
Name: fare, dtype: float64
pclass type(s) associated with deck       :
deck
A           [1]
B           [1]
C           [1]
D        [2, 1]
E     [1, 2, 3]
F        [2, 3]
G           [3]
Name: pclass, dtype: object
Passenger count linked with pclass by deck:
pclass   1  2  3
deck
A       15  0  0
B       47  0  0
C       59  0  0
D       29  4  0
E       25  4  3
F        0  8  5
G        0  0  4
```

```
The median fare specific to pclass by deck:
pclass       1       2         3
deck
A       35.500   0.000   0.00000
B       80.000   0.000   0.00000
C       83.475   0.000   0.00000
D       75.250  13.000   0.00000
E       55.000  11.425  12.47500
F        0.000  26.000   7.65000
G        0.000   0.000  13.58125
```

In [54]:

```
'''

We will use dfavg(df, col, metric='sce') to impute the missing values
under column `age`. See the Jupyter notebook on functional testing of
dfavg() using toy data.

The required additional arguments based on the heuristics devised above
for the `sce` metric in this use case are as follows.
1. sce_val_ls=[5., 30.]

2. sce_col_ls=[['who'], ['who']]

3. sce_key_ls=[['child'],['child']]

4. sce_rel_ls=[['=='],['!=']]

'''

res = dfz.dfavg(
                col='age', metric='sce',
                sce_val_ls=[5., 30.],
                sce_col_ls=[['who'], ['who']],
                sce_key_ls=[['child'],['child']],
                sce_rel_ls=[['=='],['!=']]
                )

print('metric: ', res[0])
imp = pd.DataFrame({'Age Null Indexes': res[1][0], 'Age Imputations': res[1][1]})
imp.head()
```

```
metric:  sce
```

Out[54]:

	Age Null Indexes	Age Imputations
0	5	30.0
1	17	30.0
2	19	30.0
3	26	30.0
4	28	30.0

In [56]:

```
'''

Now let's use dfavg(df, col, metric='sce') to impute the missing values
under column `deck`.

The required additional arguments based on the heuristics devised above
for the `sce` metric in this use case are as follows.

1. sce_val_ls=['A', 'B', 'C', 'D', 'D', 'E', 'E', 'E', 'F', 'F', 'G']

2. sce_col_ls=[
               ['pclass', 'fare'],
               ['pclass', 'fare', 'fare'],
               ['pclass', 'fare'],
               ['pclass', 'fare', 'fare'],
               ['pclass', 'fare', 'fare'],
               ['pclass', 'fare', 'fare'],
               ['pclass', 'fare'],
               ['pclass', 'fare'],
               ['pclass', 'fare'],
               ['pclass', 'fare'],
               ['pclass', 'fare'],
              ]

3. sce_key_ls=[
```

```
                 [1, 50],
                 [1, 80, 83],
                 [1, 83],
                 [1, 60, 80],
                 [2, 13, 20],
                 [1, 50, 60],
                 [2, 13],
                 [3, 11],
                 [2, 20],
                 [3, 11],
                 [3, 13],
                ]

4. sce_rel_ls=[
                 ['==', '>'],
                 ['==', '<=', '>'],
                 ['==', '<='],
                 ['==', '<=', '>'],
                 ['==', '<=', '>='],
                 ['==', '<', '>'],
                 ['==', '>'],
                 ['==', '<='],
                 ['==', '<'],
                 ['==', '>'],
                 ['==', '<'],
                ]

'''

res = dfz.dfavg(
                 col='deck', metric='sce',
                 sce_val_ls=['A', 'B', 'C', 'D', 'E', 'E', 'E', 'F', 'F', 'G'],
                 sce_col_ls=[
                 ['pclass', 'fare'],
                 ['pclass', 'fare', 'fare'],
                 ['pclass', 'fare'],
                 ['pclass', 'fare', 'fare'],
                 ['pclass', 'fare', 'fare'],
                 ['pclass', 'fare'],
                 ['pclass', 'fare'],
                 ['pclass', 'fare'],
                 ['pclass', 'fare'],
                 ['pclass', 'fare'],
                 ],
                 sce_key_ls=[
                 [1, 50],
                 [1, 80, 83],
                 [1, 83],
                 [1, 60, 80],
                 [1, 50, 60],
                 [2, 13],
                 [3, 11],
                 [2, 20],
                 [3, 11],
                 [3, 13],
                 ],
                 sce_rel_ls=[
                 ['==', '>'],
                 ['==', '<=', '>'],
                 ['==', '<='],
                 ['==', '<=', '>'],
                 ['==', '<', '>'],
                 ['==', '>'],
                 ['==', '<='],
                 ['==', '<'],
                 ['==', '>'],
                 ['==', '<'],
                 ]
                 )

print('metric: ', res[0])
imp = pd.DataFrame({'Deck Null Indexes': res[1][0], 'Deck Imputations': res[1][1]})
imp.head(10)
```

```
metric:  sce
```

Out[56]:

	Deck Null Indexes	Deck Imputations
0	0	F
1	2	F

2	4	F
3	5	F
4	7	G
5	8	E
6	9	F
7	12	F
8	13	G
9	14	F

In [58]:

```
'''

The relationship among `pclass`, `fare`, and `deck` is not easy to glean to infer
heuristics for imputation. Let's now use metric='mle' in dfavg() to suggest
missing value substitutions in the `deck` column.

Note:
- Specify that the `deck` column is categorical (target_type='cat'), otherwise
  it will be treated as numerical.
- A large sample size for this method is recommended.

Results below indicate that missing values cannot be estimated using the machine
learning estimation (MLE)/ Random Forest method, given the `deck' column class
counts.

The fact that the number of splits (n_splits) cannot be greater than the number of
samples  (n_samples) for the underlying machine learning method drives home the
importance of a larger sample that comprises at least 5 examples per class. In the
given data set, Deck 'G' has only 4 examples.

'''

dfz.dfavg(col='deck', target_type='cat', mle_inp_ls=['pclass', 'fare'],
          metric='mle', plot_imp=True)
```

```
df columns: ['age', 'sex', 'who', 'sibsp', 'parch', 'embarked', 'pclass', 'deck', 'fare', 'survived']
input_cols: ['pclass', 'fare']
target_col: deck
target_typ: cat
dropna    : True
idx_ls    : [0, 2, 4, 5, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 25, 26, 28, 29, 30, 32, 33, 3
4, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 56, 57, 58, 59, 60, 63, 64, 65,
67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 89, 90, 91, 93, 94, 95, 98,
99, 100, 101, 103, 104, 105, 106, 107, 108, 109, 111, 112, 113, 114, 115, 116, 117, 119, 120, 121, 122, 12
5, 126, 127, 129, 130, 131, 132, 133, 134, 135, 138, 140, 141, 142, 143, 144, 145, 146, 147, 149, 150, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 167, 168, 169, 171, 172, 173, 175, 176, 17
8, 179, 180, 181, 182, 184, 186, 187, 188, 189, 190, 191, 192, 196, 197, 198, 199, 200, 201, 202, 203, 204,
206, 207, 208, 210, 211, 212, 213, 214, 216, 217, 219, 220, 221, 222, 223, 225, 226, 227, 228, 229, 231, 23
2, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 246, 247, 249, 250, 253, 254, 255, 256, 258,
259, 260, 261, 264, 265, 266, 267, 270, 271, 272, 274, 276, 277, 278, 279, 280, 281, 282, 283, 285, 286, 28
7, 288, 289, 290, 293, 294, 295, 296, 300, 301, 302, 304, 306, 308, 312, 313, 314, 315, 316, 317, 320, 321,
322, 323, 324, 326, 328, 330, 333, 334, 335, 338, 339, 342, 343, 344, 346, 347, 348, 349, 350, 352, 353, 35
4, 355, 357, 358, 359, 360, 361, 362, 363, 364, 365, 367, 368, 371, 372, 373, 374, 375, 376, 378, 379, 380,
381, 382, 383, 384, 385, 386, 387, 388, 389, 391, 392, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 40
5, 406, 407, 408, 409, 410, 411, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,
428, 431, 432, 433, 436, 437, 439, 440, 441, 442, 443, 444, 446, 447, 448, 450, 451, 454, 455, 458, 459, 46
1, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 474, 476, 477, 478, 479, 480, 481, 482, 483, 485, 488,
489, 490, 491, 493, 494, 495, 497, 499, 500, 501, 502, 503, 506, 507, 508, 509, 510, 511, 513, 514, 517, 51
8, 519, 521, 522, 524, 525, 526, 528, 529, 530, 531, 532, 533, 534, 535, 537, 538, 541, 542, 543, 545, 546,
547, 548, 549, 551, 552, 553, 554, 555, 557, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 57
3, 574, 575, 576, 578, 579, 580, 582, 584, 586, 588, 589, 590, 592, 593, 594, 595, 596, 597, 598, 600, 601,
602, 603, 604, 605, 606, 607, 608, 610, 611, 612, 613, 614, 615, 616, 617, 619, 620, 622, 623, 624, 626, 62
8, 629, 631, 633, 634, 635, 636, 637, 638, 639, 640, 642, 643, 644, 646, 648, 649, 650, 651, 652, 653, 654,
655, 656, 657, 658, 660, 661, 663, 664, 665, 666, 667, 668, 670, 672, 673, 674, 675, 676, 677, 678, 680, 68
2, 683, 684, 685, 686, 687, 688, 691, 692, 693, 694, 695, 696, 697, 702, 703, 704, 705, 706, 708, 709, 713,
714, 718, 719, 720, 721, 722, 723, 725, 726, 727, 728, 729, 731, 732, 733, 734, 735, 736, 738, 739, 743, 74
4, 746, 747, 749, 750, 752, 753, 754, 755, 756, 757, 758, 760, 761, 762, 764, 766, 767, 768, 769, 770, 771,
773, 774, 775, 777, 778, 780, 783, 784, 785, 786, 787, 788, 790, 791, 792, 793, 794, 795, 797, 798, 799, 80
0, 801, 803, 804, 805, 807, 808, 810, 811, 812, 813, 814, 816, 817, 818, 819, 821, 822, 824, 825, 826, 827,
828, 830, 831, 832, 833, 834, 836, 837, 838, 840, 841, 842, 843, 844, 845, 846, 847, 848, 850, 851, 852, 85
4, 855, 856, 858, 859, 860, 861, 863, 864, 865, 866, 868, 869, 870, 873, 874, 875, 876, 877, 878, 880, 881,
882, 883, 884, 885, 886, 888, 890]
model flag: 0
*** Performing sanity checks on input columns before encoding categorical features...
```

```
*** Encoding input column(s)...
enc_type: cat
*** Warning! Perform encoding only after null values in the dataframe have been treated.
*** Warning! Since no columns are specified, encoding is proceeding autonomously.
*** Since the target column is suspected of being categorical, performing sanity checks before encoding...
*** Proceeding to check class membership count...
*** Warning! Found at least 1 class with membership count below the required minimum of 5 per class.
*** Attempting to convert column 'deck' to numeric dtype...
An exception of type ValueError occurred (Unable to parse string "C" at position 0)...Skipping conversion o
f column 'deck' to numeric dtype...
*** Attempting to convert column 'deck' to datetime dtype before converting to numeric dtype...
An exception of type DateParseError occurred (Unknown datetime string format, unable to parse: C, at positi
on 0)...Skipping conversion of column 'deck' to datetime dtype...
*** Failed to convert target column to either numeric or datetime dtype. Missing values cannot be estimated
using the MLE/RF method.
```

Out[58]:

```
metric              mle
result    ([], [], [])
dtype: object
```

In [91]:

```python
'''

For testing purposes, let's replace a couple of pclass=3, deck='E' entries (last
two) with deck='G' to boost the class-G count.

'''

g_filter = dfz[(dfz['pclass'] == 3) & (dfz['deck'].notna()) & (dfz['deck'] == 'E')]
g_indexes= g_filter.index.tolist()
for i in g_indexes[-2:]:
    dfz.loc[i, 'deck'] = 'G'
```

In [87]:

```python
'''

Re-run dfavg() with metric=`mle` after making the replacements above.

This time, the dfavg(metric='mle') algoritm runs since the minimum class
requirement is fulfilled. The results, for example, suggest substituting `E`
(with a model accuracy score of ~0.81) for the missing value at row index 2.
This is in contrast to the user-directed recommendation (which suggested `F`)
from dfavg(metric='sce') above.

'''

res = dfz.dfavg(col='deck', target_type='cat', mle_inp_ls=['pclass', 'fare'],
                metric='mle', plot_imp=True)

print('metric: ', res[0])
imp = pd.DataFrame({'Deck Null Indexes': res[1][0], 'Deck Imputations': res[1][1]})
imp.head(10)
```

```
df columns: ['age', 'sex', 'who', 'sibsp', 'parch', 'embarked', 'pclass', 'deck', 'fare', 'survived']
input_cols: ['pclass', 'fare']
target_col: deck
target_typ: cat
dropna    : True
idx_ls    : [0, 2, 4, 5, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 25, 26, 28, 29, 30, 32, 33, 3
4, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 56, 57, 58, 59, 60, 63, 64, 65,
67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 89, 90, 91, 93, 94, 95, 98,
99, 100, 101, 103, 104, 105, 106, 107, 108, 109, 111, 112, 113, 114, 115, 116, 117, 119, 120, 121, 122, 12
5, 126, 127, 129, 130, 131, 132, 133, 134, 135, 138, 140, 141, 142, 143, 144, 145, 146, 147, 149, 150, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 167, 168, 169, 171, 172, 173, 175, 176, 17
8, 179, 180, 181, 182, 184, 186, 187, 188, 189, 190, 191, 192, 196, 197, 198, 199, 200, 201, 202, 203, 204,
206, 207, 208, 210, 211, 212, 213, 214, 216, 217, 219, 220, 221, 222, 223, 225, 226, 227, 228, 229, 231, 23
2, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 246, 247, 249, 250, 253, 254, 255, 256, 258,
259, 260, 261, 264, 265, 266, 267, 270, 271, 272, 274, 276, 277, 278, 279, 280, 281, 282, 283, 285, 286, 28
7, 288, 289, 290, 293, 294, 295, 296, 300, 301, 302, 304, 306, 308, 312, 313, 314, 315, 316, 317, 320, 321,
322, 323, 324, 326, 328, 330, 333, 334, 335, 338, 339, 342, 343, 344, 346, 347, 348, 349, 350, 352, 353, 35
4, 355, 357, 358, 359, 360, 361, 362, 363, 364, 365, 367, 368, 371, 372, 373, 374, 375, 376, 378, 379, 380,
381, 382, 383, 384, 385, 386, 387, 388, 389, 391, 392, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 40
5, 406, 407, 408, 409, 410, 411, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,
428, 431, 432, 433, 436, 437, 439, 440, 441, 442, 443, 444, 446, 447, 448, 450, 451, 454, 455, 458, 459, 46
1, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 474, 476, 477, 478, 479, 480, 481, 482, 483, 485, 488,
489, 490, 491, 493, 494, 495, 497, 499, 500, 501, 502, 503, 506, 507, 508, 509, 510, 511, 513, 514, 517, 51
8, 519, 521, 522, 524, 525, 526, 528, 529, 530, 531, 532, 533, 534, 535, 537, 538, 541, 542, 543, 545, 546,
547, 548, 549, 551, 552, 553, 554, 555, 557, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 57
3, 574, 575, 576, 578, 579, 580, 582, 584, 586, 588, 589, 590, 592, 593, 594, 595, 596, 597, 598, 600, 601,
602, 603, 604, 605, 606, 607, 608, 610, 611, 612, 613, 614, 615, 616, 617, 619, 620, 622, 623, 624, 626, 62
8, 629, 631, 633, 634, 635, 636, 637, 638, 639, 640, 642, 643, 644, 646, 648, 649, 650, 651, 652, 653, 654,
```

```
655, 656, 657, 658, 660, 661, 663, 664, 665, 666, 667, 668, 670, 672, 673, 674, 675, 676, 677, 678, 680, 68
2, 683, 684, 685, 686, 687, 688, 691, 692, 693, 694, 695, 696, 697, 702, 703, 704, 705, 706, 708, 709, 713,
714, 718, 719, 720, 721, 722, 723, 725, 726, 727, 728, 729, 731, 732, 733, 734, 735, 736, 738, 739, 743, 74
4, 746, 747, 749, 750, 752, 753, 754, 755, 756, 757, 758, 760, 761, 762, 764, 766, 767, 768, 769, 770, 771,
773, 774, 775, 777, 778, 780, 783, 784, 785, 786, 787, 788, 790, 791, 792, 793, 794, 795, 797, 798, 799, 80
0, 801, 803, 804, 805, 807, 808, 810, 811, 812, 813, 814, 816, 817, 818, 819, 821, 822, 824, 825, 826, 827,
828, 830, 831, 832, 833, 834, 836, 837, 838, 840, 841, 842, 843, 844, 845, 846, 847, 848, 850, 851, 852, 85
4, 855, 856, 858, 859, 860, 861, 863, 864, 865, 866, 868, 869, 870, 873, 874, 875, 876, 877, 878, 880, 881,
882, 883, 884, 885, 886, 888, 890]
model flag: 0
*** Performing sanity checks on input columns before encoding categorical features...
*** Encoding input column(s)...
enc_type: cat
*** Warning! Perform encoding only after null values in the dataframe have been treated.
*** Warning! Since no columns are specified, encoding is proceeding autonomously.
*** Since the target column is suspected of being categorical, performing sanity checks before encoding...
*** Proceeding to check class membership count...
*** Encoding target column...
enc_type: cat
*** Warning! Perform encoding only after null values in the dataframe have been treated.
Generating a mask comprising a list of 35 strings.
*** Warning! Since no columns are specified, encoding is proceeding autonomously.
*** Proceeding with category encoding of col 'deck'...
*** Done!
*** Row count has dropped from 891 to 203 as a result of null removal.
model flag: 0
model     : classifier
metric    : dis_metric
*** Warning: No index is fixed. Target index is assumed to exist outside the index list considered for this
operation.
model 00 performance: 0.805 combination: ['pclass', 'fare']
model 01 performance: 0.317 combination: ['pclass']
model 02 performance: 0.805 combination: ['fare']
Top predtr: fare | Importance: 0.902
*** Proceeding with missing value prediction for the target column...
metric:  mle
```

Out[87]:

	Deck Null Indexes	Deck Imputations
0	0	F
1	2	E
2	4	E
3	5	E
4	7	F
5	8	G
6	9	F
7	12	E
8	13	F
9	14	F